International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Vol. 4, Issue 2, Apr 2014, 185-196

TRANS STELLAR
• Journal Publications • Research Consultancy

HYBRID COFI ALGORITHM FOR DATA MINING

AMOL YADAV[1], VILAS JADHAV[2], ANKUSH PAWAR[3] & MOHIL PANDEY[4]
[1,2,4]Department of Computer Engineering, MGMCET, NAVI Mumbai, Maharashtra, India
[3]Department of Computer Engineering, DRIEMS, Odisha, India

ABSTRACT

Earlier, there was a need of mining algorithm for dealing with the transactional datasets. This need result into the development of a new algorithm called Association rule mining algorithm. But now a days this existing rule mining algorithm faces many problems, when it deals with the mining of hefty transactional datasets. Some of these major problems are:1) there was a need of repetitive I/O disc scan, 2) heavy transactional datasets involves huge computations during the candidacy generation, 3) it was highly memory dependent.

This paper present the parallel implementation of our frequent item set mining algorithm, COFI, using a hybrid approach, where efficiency is achieved by applying four new techniques. 1) Mining here is done using a compact memory based data structures, 2) This algorithm generates a small independent tree which summarizes co-occurrence, 3) It uses a pruning technique which results in reducing of search space drastically & 4) It uses a simple & non recursive mining process that reduces the overall memory requirements.

KEYWORDS: Apriori Algorithm, Graphic Processing Unit (GPU), Open Computing Language (Open CP), Frequent Pattern Tree, Co Occurrence Frequent Item Trees, FP-Growth Algorithm

INTRODUCTION

Every data mining research area goes through a frequent pattern discovery process which finds the sets of items that occurs together more than a given threshold value which is defined by the decision maker. Market basket analysis is one of the application domains which go through the pattern discovery process. In a case where a threshold defined by decision maker is low & given number of frequent pattern "explodes", the discovery of sets of items that occurs together becomes hard due to reason such as 1) There is huge memory dependencies among the item sets, 2) It requires huge search space, 3) It also requires heavy I/O

In order to reduce the dependencies of heavy search space a technique called clever pruning was used. In this paper we introduce a new method for discovery of frequent pattern item sets which is totally based on the concept of COFI. The technique used in COFI algorithm saves the memory space requirements. So basically, in this algorithm based on a memory structure small trees are constructed which is called as FP trees. That is, here we introduce a new COFI tree structure & a disk based data structure, construction of inverted matrix which replaces the memory based FP tree structure and thereby significantly scales the frequent pattern mining process.

Problem Statement

Market basket analysis application suffers from the problem of mining of association rules. The problem here is to find association between items or sets of items in the given transaction data. There are various other applications such as

recommender system, diagnosis system, decision support, telecommunication system where it uses association rule mining. Formally, as defined in [3], the problem is stated as follows: Let I = fi1; i2; be a set of literals, called items and m is considered the dimensionality of the problem. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. A unique identifier TID is given to each transaction. A transaction T is said to contain X, a set of items in I, if $X \subseteq T$. An association rule is an implication of the form "$X \Rightarrow Y$", where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \phi$. An item set X is said to be large or frequent if its support s is greater or equal than a given minimum support thresholdσ. An item set X satisfies a constraint C if and only if C(X) is true. The rule $X \Rightarrow Y$ has a support s in the transaction set D if s% of the transactions in D contain $X \cup Y$. The rule $X \Rightarrow Y$ has a confidence c in the transaction set D if c% of transactions in D that contain X also contain Y. So basically the problem here consists of generating the rules that have a support and confidence greater than a given threshold. These rules are called strong rules. This association-mining task can be broken into two steps:

- A step for finding all frequent k-item sets known for its extreme I/O scan expense, and the massive computational costs;
- A straightforward step for generating strong rules.

RELATED WORK

Apriori Algorithm

Apriori Algorithm was proposed by R. Agrawal and R. Srikant in 1994 [2]. Its goal is to find those implicit and non-trivial frequent data patterns of interest from the data set or database. Whether a pattern is frequent or not is determined by a predefined threshold. A frequent pattern must occur more often in the database than, if not as often as, what the threshold defines. The Apriori method starts from finding frequent patterns with only one item. Two frequent patterns can be merged into a higher-level candidate pattern with more items. Then, each newly-generated candidate pattern is verified by checking whether its occurrence count exceeds the specified threshold or not. This step requires database scanning. The candidate generation and verification process repeats until no more new patterns can be generated.

Graphic Processing Unit (GPU)

A GPU, a microprocessor for image processing, helps the CPU for graphics processing. There are hundreds or even thousands of computing units in a GPU. Each computing unit is like a simplified core of CPU. General-purpose computing on graphics processing units (GPGPU) was proposed to use a GPU as a CPU to provide non-graphic computing capabilities. The current GPGPU technologies include Open CL and CUDA (Compute Unified Device Architecture). Since a GPU core is slower than a CPU core, it is normally assigned simple computing functions. The number of GPU cores is more than that of CPU cores, so GPU is suitable for parallel computing.

Open Computing Language (Open CL)

The preliminary work of Open CL was finished by AMD, IBM, Intel, and NVIDIA while it was initially developed by Apple Inc. The draft was submitted to the Khronos Group by the Apple Inc and then the GPGPU Khronos working group was established on June 16, 2008. Then Open CL 1.1 was released on June 14, 2010.

Open CL is a framework which allows C program development in heterogeneous platforms; that is, the framework can be applied in any system which is composed of different CPUs, GPUs, and other computing platforms. It is able to perform in different operating systems as long as the Open CL library is installed. The CPU and GPU can then

communicate with each other and work together by applying the appropriate C++ file for CPU and Kernel file for Open CL on GPUs to perform parallel computation with GPU.

Parallel Frequent Patterns Mining Algorithm on GPU

In 2010, Zhou et al. [16] proposed an Open CL GPGPU frequent pattern mining algorithm GPU-FPM. It is a coarse grain parallel Apriori algorithm using Tid tables. In the method, the Tid table entries of the items for each candidate pattern are assigned to one computing thread for comparison. For example, when the number of threads is 1024, it processes 2345 1024 candidate patterns at the same time with each thread responsible for one candidate pattern verification. This method is suitable when using fast computation units. Since each candidate pattern may have different number of entries to process, some threads may need to wait for other threads to finish their tasks.

FREQUENT PATTERN TREE: DESIGN AND CONSTRUCTION

In general there are two main stages in COFI tree approach. Stage one is to build a modified Frequent Pattern tree. Stage two consists of repetitive building of small data structures, the actual mining for these data structures, and t heir release.

Construction of the Frequent Pattern Tree

This stage aims to build the compact data structure called Frequent Pattern Tree [11]. This construction requires two phases, where each phase needs a full I/O scan of the dataset. In its initial scan, database identifies the frequent 1-itemsets with the goal is to generate an ordered list of frequent items that would be used when building the tree in the second phase. In its starting phase it enumerates the items that appear in the transactions. After enumeration these items (i.e. after reading the whole dataset), infrequent items with a support less than the support threshold are weeded out and the remaining frequent items are sorted by their frequency.

Table 1: Transactional Database

T1	A	G	D	C	B
T2	B	C	H	E	D
T3	B	D	E	A	M
T4	C	E	F	A	N
T5	A	B	N	O	P
T6	A	C	Q	R	G
T7	A	C	H	I	G
T8	L	E	F	K	B
T9	A	F	M	N	O
T10	C	F	P	G	R
T11	A	D	B	H	I
T12	D	E	B	K	L
T13	M	D	C	G	O
T14	C	F	P	Q	J
T15	B	D	E	F	I
T16	J	E	B	A	D
T17	A	K	E	F	C
T18	C	D	L	B	A

This list is organized in a table, called header table, where the items and their respective support are stored along with pointers to the first occurrence of the item in the frequent pattern tree. Phase 2 would construct a frequent pattern tree.

Phase 2 of constructing the Frequent Pattern tree structure is the actual building of this compact tree. This phase requires a second complete I/O scan from the dataset. For each transaction read, only the set of frequent items present in the header table is collected and sorted in descending order according to their frequency.

These sorted transaction items are used in constructing the FP-Trees as follows: for the first item on the sorted transactional dataset, check if it exists as one of the children of the root. If it exists then increment the support for this node. Otherwise, add a new node for this item as a child for the root node with 1 as support.

Step 1

Item	Counter	Item	Counter
A	11	N	3
B	10	O	3
C	10	P	3
D	9	Q	2
G	4	R	2
E	8	I	3
H	3	K	3
F	7	L	3
M	3	J	3

Step 2

Items	Counter
A	11
B	10
C	10
D	9
E	8
F	7

Step 3

Items	Counter
F	7
E	8
D	9
C	10
B	10
A	11

Figure 1: Steps of Phase 1

Then, consider the current item node as the new temporary root and repeat the same procedure with the next item on the sorted transaction. During the process of adding any new item-node to the FP-Tree, a link is maintained between this item-node in the tree and its entry in the header table. The header table holds as one pointer per item that points to the first occurrences of this item in the FP-Tree structure.

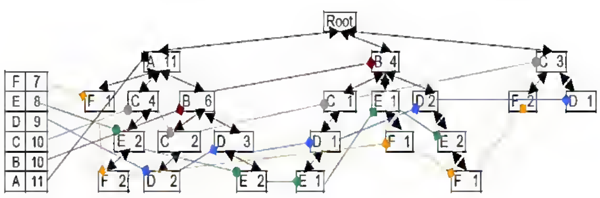

Figure 2: Frequent Pattern Tree

Illustrative Example

For illustration, we use an example with the transactions shown in Table 1. Let the minimum support threshold be set to 4. Phase 1 starts by accumulating the support for all items that occur in the transactions. Step 2 of phase1 removes all non-frequent items, in our example (G, H, I, J, K, L, M, N,O, P, Q and R), leaving only the frequent items (A, B, C, D, E, and F). Finally all frequent items are sorted according to their support to generate the sorted frequent 1-itemset. This last step ends phase 1 in Figure 1 of the COFI-tree algorithm and starts the second phase. In phase 2, the first transaction (A, G, D, C, B) is filtered to consider only the frequent items that occur in the header table (i.e. A, D, C and B). This frequent list is sorted according to the items' supports (A, B, C and D). This ordered transaction generates the first path of the FP-Tree with all item-node support initially equal to 1. A link is established between each item-node in the tree and its corresponding item entry in the header table. The same procedure is executed for the second transaction (B, C, H, E, and D), which yields a sorted frequent item list (B, C, D, E) that forms the second path of the FP-Tree. Transaction 3 (B, D, E, A, and M) yields the sorted frequent item list (A, B, D, E) that shares the same prefix (A, B) with an existing path on the tree. Item-nodes (A and B) support is incremented by 1 making the support of (A) and (B) equal to 2 and a new sub-path is created with the remaining items on the list (D, E) all with support equal to 1. The same process occurs for all transactions until we build the FP-Tree for the transactions given in Table 1. Figure 2 shows the result of the tree building process. Notice that in our tree structure, contrary to the original FP-tree [11], our links are bi-directional. This, and other differences presented later, is used by our mining algorithm.

COFI PRUNING, MINING AND CONSTRUCTION

Steps to be followed in COFI Algorithm

- Generate vertical transaction matrix
- Sort horizontal transaction matrix
- Generate inverted matrix
- Generate Individual COFI Tree for each item in inverted matrix
- Mine each COFI tree to generate frequent item sets

Pruning the COFI Trees

Pruning can be done after building a tree or, even better, while building it. We opted for pruning on the fly since the overhead is minimal but the consequences are drastic reduction in memory requirements. We will discuss the pruning idea, and then present the building algorithm that considers the pruning on the fly. In this section we are introducing a new anti-monotone property called global frequent/local non-frequent property. This property is similar to the Apriori one in the sense that it eliminates at the ith level all non-frequent items that will not participate in the (i+1) level of candidate item sets generation. The difference between the two properties is that we extended our property to eliminate also frequent items which are among the i-item set and we are sure that they will not participate in the (i+1) candidate set. The Apriori property states that all nonempty subsets of a frequent item set must also be frequent. An example is given later in this section to illustrate both properties. In our approach, we are trying to find all frequent patterns with respect to one frequent item, which is the base item of the tested COFI tree. We already know that all items that participate in the creation of the COFI-tree are frequent with respect to the global transaction database, but that does not mean that they are also locally

frequent with respect to the based item in the COFI-tree. The global frequent/local non-frequent property states that all nonempty subsets of a frequent item set with respect to the item A of the A-COFI-tree, must also be frequent with respect to item A. For each frequent item A, we traverse the FP-Tree to find all frequent items that occur with A in at least one transaction (or branch in the FP-Tree) with their number of occurrences. All items that are locally frequent with item A will participate in building the A-COFI-tree, other global frequent items, locally non-frequent items will not participate in the creation of the A-COFI-tree.

Construction of the Co Occurrence Frequent Item Trees

The small COFI-trees we build are similar to the conditional FP-Trees [11] in general in the sense that they have a header with ordered frequent items and horizontal pointers pointing to a succession of nodes containing the same frequent item, and the prefix tree per se with paths representing sub-transactions. However, the COFI-trees have bidirectional links in the tree allowing bottom-up scanning as well, and the nodes contain not only the item label and a frequency counter, but also a participation counter as explained later in this section. The COFI-tree for a given frequent item x contains only nodes labeled with items that are more frequent than or as frequent as x. To illustrate the idea of the COFI-trees, we will explain step by step the process of creating COFI-trees for the FP Tree of Figure 2. With our example, the first Co-Occurrence Frequent Item tree is built for item F as it is the least frequent item in the header table. In this tree for F, all frequent items, which are more frequent than F, and share transactions with F, participate in building the tree. This can be found by following the chain of item F in the FP-Tree structure.

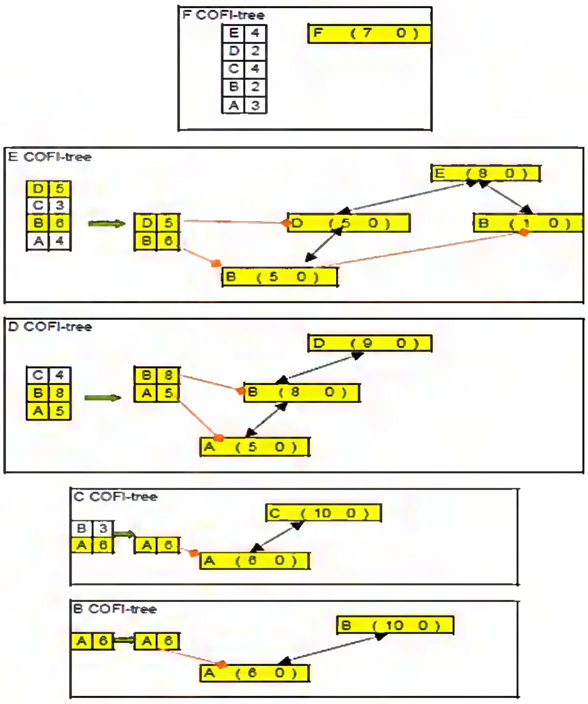

Figure 3: COFI Trees

The F-COFI-tree starts with the root node containing the item in question, and then a scan of part of the FP-Tree is applied following the chain of the F item in the FP-Tree. The first branch FA has frequency of 1, as the frequency of the branch is the frequency of the test item, which is F. The goal of this traversal is to count the frequency of each frequent item with respect to item F. By doing so we can find that item E occurs 4 times, D occurs 2 times, C occurs 4 times,

B 2 times, and A 3 times, by applying the anti monotone constraint property we can predict that item F will never appear in any frequent pattern except itself. Consequently there will be no need to continue building the F-COFI-tree. The next frequent item to test is E. The same process is done to compute the frequency of each frequent item with respect to item E. From this we can find that only two globally frequent items are also locally frequent which are (D: 5 and B: 6). For each sub-transaction or branch in the FP-Tree containing item E with other locally frequent items that are more frequent than E which are parent nodes of E, a branch is formed starting from the root node E. the support of this branch is equal to the support of the E node in its corresponding branch in FP-Tree.

If multiple frequent items share the same prefix, they are merged into one branch and a counter for each node of the tree is adjusted accordingly. Figure 3 illustrates all COFI-trees for frequent items of Figure 2. In Figure 3, the rectangle nodes are nodes from the tree with an item label and two counters. The first counter is a support-count for that node while the second counter, called participation-count, is initialized to 0 and is used by the mining algorithm discussed later, a horizontal link which points to the next node that has the same item name in the tree, and a bi-directional vertical link that links a child node with its parent and a parent with its child.

The bi directional pointers facilitate the mining process by making the traversal of the tree easier. The squares are actually cells from the header table as with the FP-Tree. This is a list made of all frequent items that participate in building the tree structure sorted in ascending order of their global support. Each entry in this list contains the item-name, item counter, and a pointer to the first node in the tree that has the same item-name.

To explain the COFI-tree building process, we will highlight the building steps for the E-COFI-tree in Figure 3. Frequent item E is read from the header table and its first location in the FP-Tree is located using the pointer in the header table. The first location of item E indicate that it shares a branch with items CA, with support = 2, since none of these items are locally frequent then only the support of the E root node is incremented by 2. The second node of item E indicates that it shares items DBA with support equals to 2 for this branch as the support of the E-item is considered the support for this branch (following the upper links for this item). Two nodes are created, for items D and B with support equals to 2, D is a child node of B, and B is a child node of E.

The third location of E indicate having EDB: 1, which shares an existing branch in the E-COFI-tree, all counters are adjusted accordingly. A new branch of EB: 1 is created as the support of E=1 for the fourth occurrences of E. The final occurrence EDB: 2 use an existing branch and only counters are adjusted. Like with FP-Trees, the header constitutes a list of all frequent items to maintain the location of first entry for each item in the COFI-tree. A link is also made for each node in the tree that points to the next location of the same item in the tree if it exists. The mining process is the last step done on the E-COFI-tree before removing it and creating the next COFI-tree for the next item in the header table.

Mining the COFI Trees

The COFI-trees of all frequent items are not constructed together. Each tree is built, mined, and then discarded before the next COFI-tree is built. The mining process is done for each tree independently with the purpose of finding all frequent k-item set patterns in which the item on the root of the tree participates. The following is our algorithm for building and mining the COFI-trees with pruning.

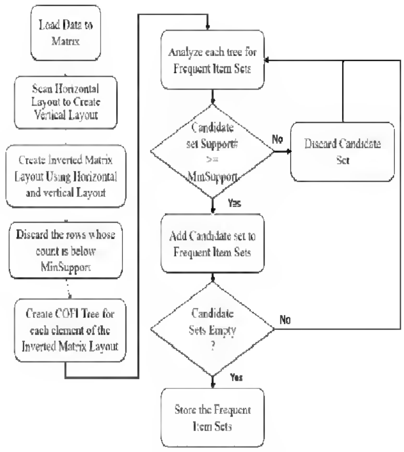

Figure 4: Flow of COFI Algorithm

Algorithm of COFI algorithm is given below:

Algorithm part 1:

Inverted Matrix (IM) Construction (2)

Input: Transactional Database (D)

Output: Inverted Matrix

METHODS

Pass I

- Scan D to identify unique items with their frequencies.
- Sort the items in ascending order of their frequency.
- Create the index part of the IM using the sorted list.

Pass II

- While there is a transaction T in the database (D)

Do

1.1 Sort the items in the transaction T into ascending order according to their frequency

1.2 while there are items si in the transaction do

1.2.1 Add an entry in its corresponding transactional array row with 2- parameters

(A) Location in index part of the IM of the next item si+1 in T null if si+1 do not exist.

(B) Location of the next empty slot in the transactional array row of si+1, null if si+1 do not exist.

1.3 Go to 1.2

2. Go to 1

Algorithm part 2:

Creating and Mining COFI-Trees (2)

Input: Inverted Matrix (IM) and a minimum support threshold

Output: Full set of frequent patterns Method:

1. Frequency Location = Apply binary search on the index part of the IM to _nd the Location of the _rst frequent item based on _.

2. While (Frequency Location < IM Size) do

2.1 A = Frequent item at location (Frequency Location)

2.2 Remove all non-locally frequent items for the frequent list of item (A)

2.3 Create a root node for the (A)-COFI-tree with both frequency-count and participation-count = 0

2.3.1 C is the path of locally frequent items in the path of item A to the root

2.3.2 Items on C form a prefix of the (A)-COFI-tree.

2.3.3 If the prefix is new then Set frequency-count= frequency of (A) node and participation count= 0 for all nodes in the path

Else

2.3.4 Adjust the frequency-count of the already exist part of the path.

2.3.5 Adjust the pointers of the Header list if needed

2.3.6 Find the next node for item A in the FP-tree and go to 2.3.1

2.4 Mine COFI-tree (A)

2.5 Release (A) COFI-tree

2.6 A = next frequent item from the header table

3. Go to 2

Function: Mine COFI-tree (A)

1. Node A = select next node

2. While there are still nodes do

2.1 D = set of nodes from node A to the root

2.2 F = node A. frequency-count-node A. participation-count

2.3 Generate all Candidate patterns X from items in D. Patterns that do not have A will be discarded.

2.4 Patterns in X that do not exist in the A-Candidate List will be added to it with frequency = F otherwise just increment their frequency with F

2.5 Increment the value of participation-count by F for all items in D

2.6 node A = select next node

3. Go to 2

4. Based on support threshold remove non-frequent patterns from A Candidate List.

CONCLUSIONS AND FUTURE WORK

We propose in this paper COFI algorithm which is based on our COFI-tree structure, in terms of memory usage it is better than the FP-Growth algorithm because it uses clever pruning technique. This Algorithm achieves this results thanks to: (1) the non recursive technique used in COFI-tree helps to generate a full set of frequent patterns during the mining process (2) The pruning method that is used leave all the locally frequent items in the COFI-tree and removes all locally non frequent patterns. The huge advantage of COFI algorithm over FP Growth is that it requires a significantly lesser memory; thereby it can mine heavy transactional databases with already available main memory which is very less as compared to FP Growth. The difference between COFI and FP-Growth is that, COFI tries to find a negotiate between a fully pattern growth approach, where as FP Growth focus on generating candidacy approach for which the Apriori algorithm is known for. During the mining process, COFI built targeted patterns and focus on generation of candidates. We have developed algorithm in order to avoid the recursion that FP-growth uses. Typically this algorithm is generated for closed item set mining and maximal item set mining. These efficient algorithms and experimental results will be compared to existing algorithms such as CHARM [17], MAFIA [6] and CLOSET [15], and will be reported in the future.

REFERENCES

1. R. Agarwal, C.Aggarwal, and V. Prasad. A tree projection algorithm for generation of frequent item sets. Parallel and distributed Computing, 2000.
2. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data, pages 207–216, Washington, D.C., May 1993.
3. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Proc. 1994 Int. Conf. Very Large Data Bases, pages 487–499, Santiago, Chile, September 1994.
4. I. Almaden. Quest synthetic data generation code http://www.almaden.ibm.com/cs/quest/syndata.html.
5. M.-L. Antonie and O. R. Za¨ıane. Text document categorization by term association. In IEEE International Conference on Data Mining, pages 19–26, December 2002.
6. C. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent item set algorithm for transactional databases. In IEEE International Conference on Data Mining (ICDM 01), April 2001.
7. M. El-Hajj and O. R. Za¨ıane. Inverted matrix: Efficient discovery of frequent items in large datasets in the context of interactive mining. In In Proc. 2003 Int'l Conf. on Data Mining and Knowledge Discovery (ACM SIGKDD), August 2003.
8. M. El-Hajj and O. R. Za¨ıane. Non recursive generation of frequent k-item sets from frequent pattern tree representations. In In Proc. of 5th International Conference on Data Warehousing and Knowledge Discovery (DaWak'2003), September 2003.

9. M. El-Hajj and O. R. Za¨ıane. Parallel association rule Mining with minimum inter-processor communication. In Fifth International Workshop on Parallel and Distributed Databases (PaDD'2003) in conjunction with the 14th Int' Conf. on Database and Expert Systems Applications DEXA2003, September 2003.

10. J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufman, San Francisco, CA, 2001.

11. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In ACMSIGMOD, Dallas, 2000.

12. J. Hipp, U. Guntzer, and G. Nakaeizadeh. Algorithms for association rule mining - a general survey and comparison. ACM SIGKDD Explorations, 2(1):58–64, June 2000.

13. J. Liu, Y. Pan, K. Wang, and J. Han. Mining frequent item sets by opportunistic projection. In Eight ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining, pages 229–238, Edmonton, Alberta, August 2002.

14. J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. Hmine: Hyper-structure mining of frequent patterns in large databases. In ICDM, pages 441–448, 2001.

15. J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed item sets. In 9th ACMSIGKDD International Conf. on Knowledge Discovery and Data Mining, July 2003.

16. O. R. Za¨ıane, J. Han, and H. Zhu. Mining recurrent items in multimedia with progressive resolution refinement. In Int. Conf. on Data Engineering (ICDE'2000), pages 461–470, San Diego, CA, February 2000.

17. M. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed item set mining. In 2nd SIAM International Conference on Data Mining, April 200.